The M-CASH Resource Reclaiming Algorithm for Identical Multiprocessor Platforms

Rodolfo Pellizzoni and Marco Caccamo
Department of Computer Science
University of Illinois at Urbana-Champaign
{*rpelliz2,mcaccamo*}*@uiuc.edu*

Abstract

Resource reclaiming schemes are applied in reservation-based real-time uniprocessor systems to support efficient reclaiming and sharing of computational resources left unused by early completing tasks, improving the response times of aperiodic and soft tasks in the presence of overruns. In this paper, we introduce algorithm M-CASH, a new efficient reclaiming scheme for multiprocessor platforms. M-CASH leverages the resource reservation approach offered by the Multiprocessor CBS server offering significant improvements. The correctness of the algorithm is formally proven and its performance is experimentally evaluated through several synthetic simulations.

1. Introduction

In most real-time systems, task feasibility is guaranteed by means of offline schedulability tests based on worst-case execution times (WCETs). While such an approach can be fine for very critical systems composed of hard real-time tasks with mostly regular execution times and precisely estimated WCETs, it can easily lead to an excessive waste of resources when task execution times exhibit a high variance. Furthermore, the hardware architecture of modern processors can negatively impact the WCET computation due to the unpredictability of elements such as caches and prefetching queues.

In many systems of interest hard real-time tasks must coexist together with soft real-time tasks, such as multimedia applications, whose temporal requirements are expressed in term of a desired quality of service. The WCET case is often rare but many times higher than the average case; for MPEG players, for example, the amount of time needed is strongly dependant on the frame type. For such soft tasks, guarantees based on average execution times are typically acceptable; however, it is required that in the case of an overrun(i.e., the soft task executes for more than it is guaranteed), the schedulability of hard tasks is not compromised. Unfortunately, common real-time schedulers including the uniprocessor optimal Earliest Deadline First (EDF) are not able to distinguish tasks based on criticality.

A general technique for limiting the effects of overruns relies on the concept of *resource reservation*: each task is assigned a fraction of the computational resources and it is handled by a *server* which ensures that the task does not require more resources than its assigned share. The scheme is akin to a two level scheduling framework: a top-level scheduler menages the execution requests produced by each server, which in turn serves the instances of its assigned task. Temporal isolation is thus guaranteed among tasks, in the sense that the behavior of one task can not compromise the schedulability of a different task as long as the total amount of reserved resources is within a proven bound. An efficient method for resource reservation under the EDF scheduler is the Constant Bandwidth Server (CBS) [2]. A problem of such algorithm is that the performance of the system is dependant on the correct allocation of resource shares; in particular, as we will show in Section 2, if one server has an early completion other servers are not able to efficiently reuse the amount of computational resources left unused. To overcome this problem, different *resource reclaiming* techniques [7, 10, 8, 11] have been proposed to augment the CBS approach with efficient reclamation and sharing of resources. Such techniques have been proven to be successful in maximizing the responsiveness of soft and aperiodic tasks, improving the usage of system resources while preserving all hard real-time constraints.

Unfortunately, all such algorithms do not easily extend to the multiprocessor case, which is becoming increasingly important. Uniprocessor systems are quickly becoming inadequate to service the high level of performance required by today's embedded systems. In fact, a general shift from unicore to multicore processors can be seen in the industry both in the general purpose and embedded domain, in an effort to increase performance while keeping chip complexity and power consumption at acceptable levels.

Research on real-time multiprocessor systems is fairly active. Although an optimal algorithm exists [3], it is typically unsuitable for most applications due to its extremely high number of preemptions. Therefore, despite being suboptimal, classic uniprocessor schedulers like EDF or Fixed Priority (FP) are still being applied to the multiprocessor case; proven schedulability conditions for hard systems have been proposed in [9, 6, 4]. In particular, in this work we will be concerned with global EDF scheduling, where all tasks are kept in a single ready queue and job are allowed to migrate among processors. The CBS server has been extended to the multiprocessor case, algorithm M-CBS, in [5]. However, to the best of our knowledge there is lack of resource reclaiming schemes for multiprocessors in literature. In this paper, we introduce M-CASH, a resource reclaiming mechanism for identical multiprocessor platforms built on top of the M-CBS algorithm. Our new scheme for multiprocessor has features similar to the CASH reclaiming algorithm [7] developed for uniprocessor platform. As we prove in Section 3, our algorithm is able to maintain temporal isolation and guarantee all hard task executions (based on the sufficient EDF schedulability bound introduced by Goossens et al. [9]) while providing efficient and effective resource reclaiming among servers.

The rest of the paper is organized as follows. In Section 2 we introduce some notation and describe the M-CBS servers and its problematics in the presence of early completing tasks. In Section 3 we introduce our new M-CASH scheme, show how it can be used to solve the resource reclaiming problem, and prove its correctness. Finally, in Section 4 we validate the effectiveness of our proposed scheme by confronting it with M-CBS through several synthetic simulations and in Section 5 we provide concluding remarks and future work.

2. M-CBS

Since our resource reclaiming scheme is based on the Multiprocessor Constant Bandwidth Server (M-CBS) first introduced by Baruah et al. in [5], we will start by describing such algorithm. Our description and proofs are different from the ones introduced in [5], being more closer to the description of the original CBS server for uniprocessors [1, 2]; this is done to introduce the reader to the notation and basic lemmas that we will use in proving the correctness of the M-CASH reclaiming scheme in Section 3.

We consider a real-time multiprocessor platform π comprised of M identical processors and N M-CBS servers $S_1, \ldots, S_N$. Each M-CBS server S_i is characterized by an ordered pair (Q_i, T_i), where Q_i is the server's *maximum budget* and T_i is the server period. The ratio $U_i = \frac{Q_i}{T_i}$ is known as the server *bandwidth* and denotes the fraction of the capacity of one processor that is assigned to the server; define $U_\pi = \sum_i = 1^N U_i$ as the system utilization and $u_\pi = \max_{1 \leq i \leq N} U_i$ as the maximum server bandwidth. Each server S_i serves a single task τ_i, which is composed by a stream of jobs τ_{ij} $(j = 1, 2, \ldots)$, each characterized by an arrival time a_{ij} and an execution time c_{ij}. Each job τ_{ij} may be further characterized by a deadline d_{ij}, but we do not require it, nor we impose any constraint on interarrival times; hence, the model is general enough to represent hard, soft, periodic or aperiodic tasks.

At each instant, an absolute deadline d_i and a budget c_i is associated with each server S_i. Each time a new job of τ_i arrives, it is enqueued in a FCFS queue held by S_i. The server is said to be *active* if its job queue is not empty, otherwise it is *idle*; whenever the server is active, the job at the top of the queue is inserted in the ready queue with deadline equal to d_i. Jobs are scheduled by global EDF. A single ready queue exists in the system; at each instant, the M higher priority (with shorter absolute deadline) jobs are scheduled for execution. The server budget and deadline are updated using the following rules:

1. Whenever the server executes, the budget c_i is decreased by the same amount. Upon reaching 0, the budget is recharged to Q_i and d_i is incremented by T_i.
2. Whenever the server transitions from idle to active at some time t, a test is executed. If $c_i < (d_i - t)U_i$, no update of deadline and budget is necessary; otherwise, c_i is recharged to Q_i and the deadline is assigned a new value: $d_i \leftarrow t + T_i$.

To ease the algorithm discussion, we now introduce some further notation. Note that from the point of view of the global scheduler, each server performs three actions: it inserts an execution request in the ready queue each time the server transitions from idle to active state; it removes the execution request when it transitions from active to idle; it postpones the deadline of its execution request once the budget is depleted. Note that postponing a deadline is effectively equal to removing the current execution request and inserting a new one. Hence, the execution requests made by a server S_i are effectively equivalent to a series of *server jobs* S_{ij} $(j = 1, 2, \ldots)$ each of which is characterized by an arrival time a_{ij}^S, that is the time at which the execution request is inserted in the ready queue, a deadline d_{ij}^S which is the server deadline for that execution request and a computation time c_{ij}^S which is the amount of time that the server executes on behalf of S_{ij} (equivalently, the amount of budget consumed by S_{ij}). It is important to note that a_{ij}^S, c_{ij}^S do not necessary correspond to a_{ij}, c_{ij}.

A clarifying example is presented in Figure 1, where a system with 2 processors comprised of two hard tasks τ_1, τ_2 and a server S_3 serving an aperiodic task τ_3 is represented. In the figure, up arrows represent arrival times while down arrows represent deadlines; furthermore, we superim-

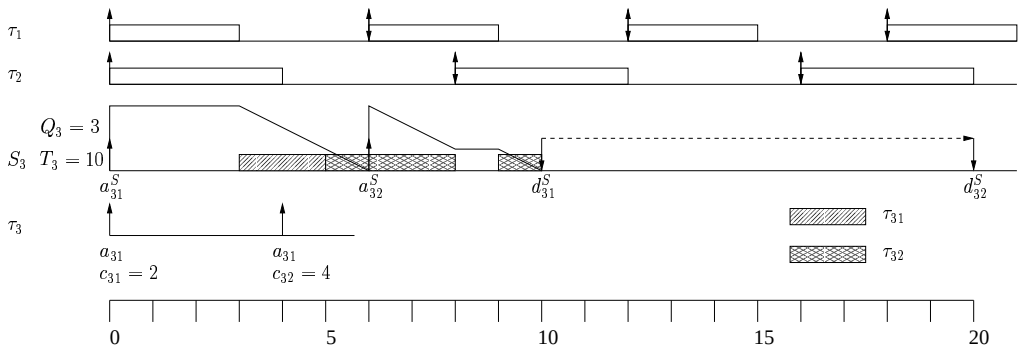


Figure 1. M-CBS Example

pose the current budget of server S_3 over its schedule and we represent the execution of jobs of τ_3 with different colors. A server job S_{31} is created at $t = 0$ when the first task job τ_{31} arrives. When τ_{32} arrives at $a_{32} = 4$, it is simply enqueued at S_3. When the budget of S_3 is consumed at time $t = 6$, a new server job S_{32} is created. Hence, note that the execution of S_{31} corresponds to the execution of both τ_{31} and part of τ_{32}.

We now prove that algorithm M-CBS is correct, in the sense that server jobs never miss their deadlines, or equivalently, whenever a server deadline d_i is reached, $c_i = 0$. Hence, the server is able to guarantee an execution time Q_i every T_i time units to its served task. We first introduce some definitions and basic lemmas.

Definition 1 (Demand function) *The demand of a server S_i, $\mathrm{dbf}_i(t_1, t_2)$ is the sum of the computation times of server jobs with arrival time greater than or equal to t_1 and deadline less than or equal to t_2:*

$$\mathrm{dbf}_i(t_1, t_2) = \sum_{j: a_{ij}^S \geq t_1, d_{ij}^S \leq t_2} c_{ij}^S$$

Lemma 1 *For each server S_i and $\forall\ t_1 \geq 0, \forall\ t_2 \geq t_1$:*

$$\mathrm{dbf}_i(t_1, t_2) \leq U_i(t_2 - t_1)$$

Proof.
Since the server job deadline is the deadline of the server while the server job is active, Definition 1 is equivalent to the demand definition in [1]; hence, the lemma follows directly from Theorem 1 in [1]. □

Definition 2 (Uniform multiprocessor) *A uniform multiprocessor platform π' is composed by M' processors $P'_1, \ldots, P'_M$. Each uniform processor is characterized by a speed s_i: a job that executes on P'_i for Δ time units completes $s_i\Delta$ units of execution. Furthermore, we define:*

$$s_{\pi'} = \max_{1 \leq i \leq M'} s_i$$

$$S_{\pi'} = \sum_{1 \leq i \leq M'} s_i$$

Note that given the above definition our previously introduced identical multiprocessor platform π is a special instance of a uniform multiprocessor platform with $s_\pi = 1, S_\pi = M$.

The correctness of M-CBS is then derived as follows: first, we introduce a suitable uniform platform π' such that the set I of server jobs generated by the M-CBS schedule is feasible on π', i.e. there exists an algorithm that feasibly schedules I on π'. The following theorem (see [9]) is then used to prove that I is schedulable by EDF on our identical multiprocessor platform π.

Theorem 2 *Consider a uniform multiprocessor system π' with $s_{\pi'} \leq 1$, and let I denote an instance of jobs (with deadlines) that is feasible on π'. Then I is feasibly scheduled by global EDF on π if:*

$$M \geq \frac{S_{\pi'} - s_{\pi'}}{1 - s_{\pi'}}$$

Lemma 3 *No server job S_{ij} misses its deadline if server S_i is assigned to a dedicated uniform processor with speed $s_i = U_i$.*

Proof.
Since the processor is dedicated, no server job can be preempted. By contradiction. Assume that a deadline is missed at time t_2. Let t_1 be the last time before t_2 such that the server transitions from idle to active state. By definition and since the processor is dedicated to the server, t_1 corresponds to the activation time of a server job S_{ij}; furthermore, all previous jobs $S_{ik}, k < j$, must have finished before t_1. Since a deadline is missed at t_2, it follows that all jobs executed in $[t_1, t_2]$ must have deadlines less than or equal to t_2 and the server continuously executes in $[t_1, t_2]$; hence, it must hold that $\frac{dbf_i(t_1,t_2)}{s_i} > t_2 - t_1$. But since $s_i = U_i$, it follows: $dbf_i(t_1, t_2) > U_i(t_2 - t_1)$, which contradicts Lemma 1. □

Theorem 4 *No server job S_{ij} executed on the identical multiprocessor platform π misses its deadline under the following conditions:*

$$u_\pi \leq 1;$$
$$U_\pi \leq M - u_\pi(M-1)$$

Proof.
Due to Lemma 3, each server S_i can be feasibly executed on a dedicated uniform processor P_i' with speed $s_i = U_i$. Hence, all servers can be feasibly executed on a uniform platform π' with $S_{\pi'} = U_\pi$ and $s_{\pi'} = u_\pi$. Since $U_\pi \leq M - u_\pi(M-1)$, it directly follows by simple algebraic manipulation that $M \geq \frac{S_{\pi'} - s_{\pi'}}{1 - s_{\pi'}}$. Therefore, we can apply Theorem 2 to the set I of all server jobs generated by π and conclude that no server job S_{ij} misses its deadline on the identical multiprocessor π. □

Note that the scheduling condition of Theorem 4 is equivalent to the schedulability bound expressed in [9] for periodic task sets scheduled by global EDF; hence, we can say that each M-CBS server requires no more bandwidth than a periodic task with utilization equal to the server bandwidth. The following theorem proves that the server is able to provide its entire reserved bandwidth to the served task.

Theorem 5 *A periodic task τ_i, with period and relative deadline equal to T_i and execution time equal to the maximum budget Q_i, can be feasibly scheduled by M-CBS server S_i under the conditions of Theorem 4.*

Proof.
Clearly $a_{i1}^S = a_{i1}$ since S_i is initially idle. Due to Theorem 4, S_{i1} will finish before d_{i1}^S; hence, since the computation time of τ_i is equal to Q_i, τ_{i1} will complete before its deadline leaving a budget $c_i = 0$. If we assume that a job τ_{ij} meets its deadline leaving zero budget, then by following the same reasoning, job τ_{ij+1} will be released at $a_{ij+1} = d_{ij}^S$ and will therefore meet its deadline as well. Hence, the theorem holds by induction. □

Theorems 4 and 5 imply that the schedulability of τ_i depends only on the parameters of the servers and not on the execution times of the other tasks in the system; hence, we conclude that the M-CBS mechanism is effective in providing temporal isolation among tasks. However, M-CBS servers are not able to efficiently reclaim unused computation time. An example is provided in Figure 2(a) for a system composed by $M = 1$ processor. The system is comprised by three servers S_1, S_2, S_3 with budgets $Q_1 = 1, Q_2 = 5, Q_3 = 3$ and period $T_1 = 4, T_2 = 10, T_3 = 12$, each serving a periodic task with period and relative deadline equal to the server period. At time $t = 6$, job τ_{21} completes one time unit earlier than the allocated budget, while job τ_{31} experiences an overrun of 1 time unit. In spite of the budget left free by job τ_{21} at $t = 6$, S_3 will postpone the deadline at time $t = 9$, forcing τ_{31} to miss its deadline. As we will see in the next section, using the reclamation mechanism introduced by M-CASH task τ_{31} can be feasibly scheduled despite the overrun[1].

3. M-CASH

The M-CASH scheme augments the M-CBS algorithm by adding a powerful reclaiming scheme; it extends the capacity sharing mechanism introduced by [7] for uniprocessor systems to the identical multiprocessor case.

The M-CASH algorithm holds a global queue of capacities (the M-CASH queue) ordered by non decreasing absolute deadline. Each time a server S_{ij} becomes idle with budget greater than zero, a new capacity $\mathrm{Cap}_{ij}(c_q, d_q)$ is inserted into the queue with the current server deadline and the remaining capacity. Each time there is a capacity at the head of the M-CASH queue and one or more servers with deadline greater than or equal to the capacity deadline are scheduled for execution, those servers consume the capacity instead of their own budget. To get the key idea of the algorithm, in Figure 2(b) we show the M-CASH scheduling for the same system of Figure 2(a). At time $t = 6$, a new capacity with value 1 and deadline 10 is inserted into the queue. This capacity is consumed by server S_3, enabling the completion of job τ_{31} within its deadline $d_{31} = 12$.

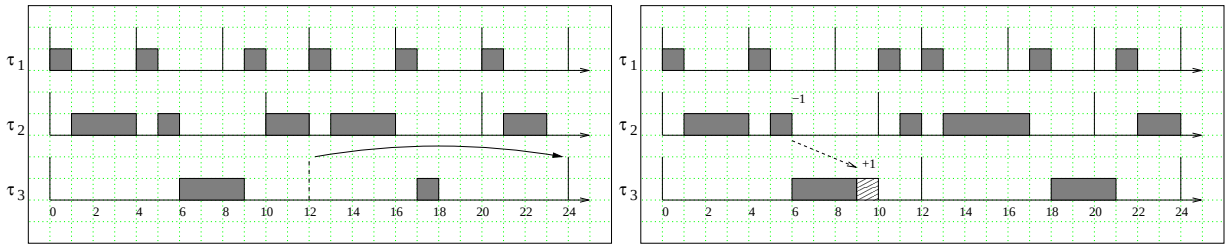
We now formally describe the rules of algorithm M-CASH. To ease the description, let us introduce some additional notation. At each time t, let E be the set of servers scheduled for execution on the identical multiprocessor platform π. Furthermore, let $V \subseteq E$ be the subset of servers executing with absolute deadline strictly less than the deadline of the capacity (c_q, d_q) at the head of the M-CASH queue:

$$V = \{S_i | S_i \text{ executing} \wedge d_i < d_q\}$$

If no capacity is present in the M-CASH queue at time t, we simply impose $V = E$. Finally, let W be the number of idle processors at time t, i.e. $W = M - \#E$, where $\#E$ denotes the cardinality of set E. The set of M-CASH rules can then be expressed as follows:

1. Whenever a server $S_i \in V$ executes, its budget c_i is decreased by the same amount. Upon reaching 0, the budget is recharged to Q_i and d_i is incremented by T_i.
2. Whenever a server S_i becomes idle and $c_i > 0$, a new capacity $\mathrm{Cap}_{ij}(c_q = c_i, d_q = d_i)$ is inserted in the capacity queue and c_i is discharged to 0.

1 While this simple example describes a uniprocessor system for the sake of simplicity, M-CAH allows to perform resource reclaiming efficiently in a multiprocessor environment.



(a) *M-CBS* **(b) *M-CASH***

Figure 2. Example overrun

3. Whenever a server S_i becomes active, c_i is recharged to Q_i and the deadline is assigned a new value: $d_i \leftarrow \max(d_i, t) + T_i$.
4. Whenever a capacity (c_q, d_q) is at the head of the M-CASH queue, the servers in $E - V$ (i.e., servers with deadline greater or equal to d_q) execute consuming the capacity instead of their own budget; furthermore, the capacity is also consumed on each of the W idle processors. Hence, in an interval of length Δ, c_q is decreased by an amount $\Delta(\#(E - V) + W) = \Delta(M - \#V)$.
5. Whenever c_q reaches 0, the capacity is removed from the queue.

Note that the M-CASH rules are an extension of the CASH rules for uniprocessor systems [7], in the sense that a capacity can only be used by servers with deadline greater or equal than the deadline of the capacity and the capacity is discharged when a processor is idle. In particular, for $M = 1$ the M-CASH algorithm reduces exactly to the CASH algorithm. Furthermore, note that from the point of view of the global scheduler a M-CASH server performs the same actions as a M-CBS server; hence, we can also abstract the execution requirements of a M-CASH server as a series of server jobs S_{ij}. However, to the contrary of a M-CBS server the computation time c_{ij}^S of a M-CASH server job S_{ij} can be greater than the the amount of budget consumed during S_{ij}, since the server can execute consuming a capacity instead of its own budget.

A comprehensive example of a M-CASH scheduling on $M = 2$ processors can be found in Figure 3. The system consists of four servers S_1, S_2, S_3, S_4 with different periods and bandwidth $U_i = 0.4$. Since $U_\pi = 1.6, u_\pi = 0.4$, the server set can be feasibly scheduled by global EDF on two processors. The job set is as follows:

- τ_1 is a hard sporadic task; the first task job arrives at $t = 0$, the second at $t = 6$ while all successive jobs arrive after the minimum interarrival time $T_1 = 5$.
- τ_2 is a firm task; its first instance τ_{21} arrives at $t = 0$ and finishes after 3 time units, inserting a capacity $\mathrm{Cap}_{21}(3, 15)$ into the M-CASH queue at time 5. The next periodic instance is skipped.
- τ_3 is a hard periodic task; τ_{31} arrives at time $t = 0$ and executes for 9 time units out of the 10 reserved by S_3.
- τ_4 is an aperiodic task. The activation times and computation times of its jobs τ_{41}, τ_{42} are shown in figure.

In the interval $[5, 6]$, the capacity $\mathrm{Cap}_{21}(3, 15)$ is at the head of the M-CASH queue and the set of scheduled server is $E = \{S_3, S_4\}$. Furthermore, note that server S_4 has to postpone its deadline at $t = 4$; therefore, all the servers in E have deadline greater than or equal to d_q and $V = \emptyset$. Hence, the capacity is decreased by $M - \#V = 2$ time units while S_3, S_4 do not consume their budget. To underline the fact that servers consume the capacity instead of their own budget, we have used a different color for portions of server job execution where a capacity is consumed. Note that at time $t = 6$ server S_1 becomes active and preempts S_3. Since $d_{12} = 11 < d_q = 15$, in the interval $[6, 7], E = \{S_1, S_4\}, V = \{S_1\}$. Therefore, the capacity is decreased by $M - \#V = 1$ time unit; S_4 uses the capacity while S_1 has to consume its own budget. The capacity is depleted at time $t = 7$ and removed from the queue. At time $t = 16$, S_3 becomes idle and a capacity $\mathrm{Cap}_{31}(2, 25)$ is inserted in the queue. Since in the interval $[16, 18], E = V = \{S_1\}, M - \#V = 1$, the capacity is depleted and removed from the queue at $t = 18$. Note that since one of the processors is left idle, the capacity is effectively wasted.

3.1. Proof of correctness

To prove that algorithm M-CASH is correct, we need to again show that server jobs never miss their deadline. We use a framework similar to the one in Section 2; however, the fact that a server can execute consuming a capacity in-

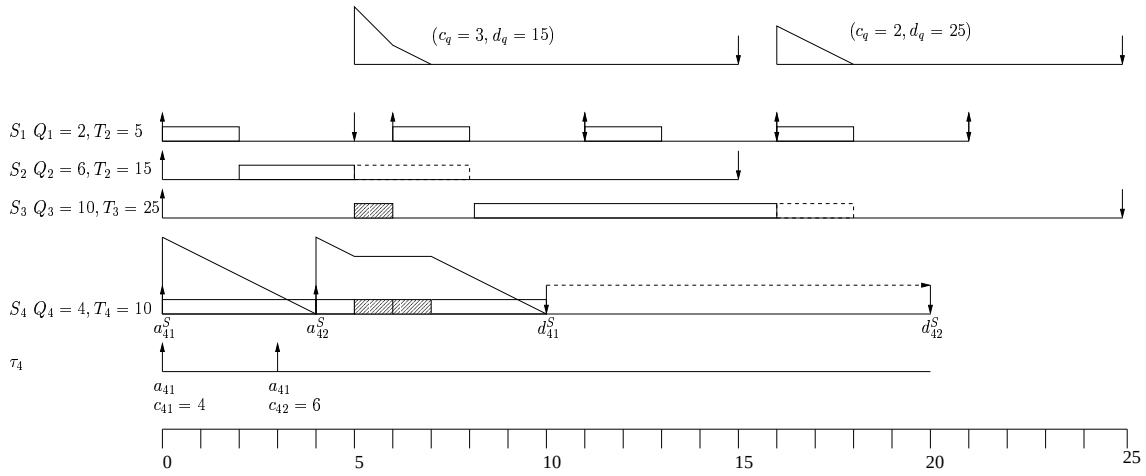


Figure 3. M-CASH Example

stead of its own budget complicates the analysis, since the demand of a server job can exceed the server's reserved bandwidth. We will therefore use the following approach: instead of considering the M-CASH scheduling of a set I of server jobs on π, we will consider the EDF scheduling of a *transformed set* of jobs $I^\star$ on the same identical multiprocessor platform π. $I^\star$ is composed by two types of jobs: a set $S^\star$ of transformed jobs $S^\star_{ij}$ that are used to represent the executions of server jobs in I where the server budget is consumed, and a set of *capacity jobs* $\mathrm{Cap}^\star$ that are used to represent the consumption of a capacity by either a server or an idle processor. In particular, to account for the fact that each capacity C_{ij} can be consumed on multiple processors and with different rates in different time intervals, multiple capacity jobs $\mathrm{Cap}^\star_{ij}$ are generated by each capacity in the M-CASH queue. Consider for example the M-CASH schedule in Figure 3 in the interval $[5, 7]$. Since capacity $\mathrm{Cap}_{21}(3, 15)$ is consumed by two servers in the interval $[5, 6]$ and one server in the interval $[6, 7]$, its consumption is represented by three capacity jobs in $I^\star$; also, the transformed job $S^\star_{42}$ executes for only 4 time units instead of the 6 time units of the original job S_{42}.

Definition 3 (M-CASH job transformation) *Consider a set I of server jobs generated by M-CASH system π. We can then define job sets $S^\star$, $\mathrm{Cap}^\star$ as follows.*

- *For each $S_{ij} \in I$, we have $S^\star_{ij}(a^\star_{ij} = a^S_{ij}, d^\star_{ij} = d^S_{ij}, c^\star_{ij} = c^S_{ij} - \Delta_{ij}\} \in S^\star$, where Δ_{ij} is the amount of time that S_{ij} executes consuming a capacity.*
- *For each capacity $\mathrm{Cap}_{ij}(c_q, d_q)$ in the M-CASH schedule of I, set $\mathrm{Cap}^\star$ is comprised by a set $\mathrm{Cap}^\star_{ij}$ formed as follows.*
  - *Let t_0 be the time when Cap_{ij} is inserted in the M-CASH queue (equivalently, the time when S_{ij} finishes). Furthermore, let t'_1 be the first time when Cap_{ij} is consumed and let t_1 be the last time such that the consumption rate is constant in interval $[t'_1, t_1]$ and equal to $K_l = M - \#V$ as defined by M-CASH Rule 4. Then $\mathrm{Cap}^\star_{ij}$ is comprised by K_1* capacity jobs *all with the same activation times $a^\star_{ij1} = t_0$, deadline $d^\star_{ij1} = d_q$ and computation time $c^\star_{ij1} = t_1 - t'_1$.*
  - *Until the capacity is depleted, for each successive interval $[t'_l, t_l]$ such that $\mathrm{Cap}_{ij}(c_q, d_q)$ is consumed at a constant rate $K_l = M - \#V$, add K_l capacity jobs to $\mathrm{Cap}^\star_{ij}$, all with the same activation time $a^\star_{ijl} = t_{l-1}$, deadline $d^\star_{ijlk} = d_q$ and computation time $c^\star_{ijlk} = t_l - t'_l$.*

Then $I^\star$, the M-CASH job transformation of I, is the union set of $S^\star$ and $\mathrm{Cap}^\star$.

To better understand how the transformed set $I^\star$ is created, consider once again Figure 3. The three capacity jobs generated by $\mathrm{Cap}_{21}(3, 15)$ are:

- for the interval $[t'_1 = 5, t_1 = 6]$ with $t_0 = 5$, two capacity jobs with activation time $a^\star_{211} = 5$, deadline $d^\star_{211} = 15$ and computation time $c^\star_{211} = 1$;
- for the interval $[t'_2 = 6, t_2 = 7]$, one capacity jobs with activation time $a^\star_{212} = 6$, deadline $d^\star_{211} = 16$ and computation time $c^\star_{211} = 1$.

The capacity $\mathrm{Cap}_{31}(2, 25)$ generates a single capacity job with activation time $a^\star_{311} = 16$, deadline $d^\star_{311} = 25$ and computation time $c^\star_{311} = 2$. Note that this capacity job does not correspond to the execution of any server in I; it is added to the capacity job set $\mathrm{Cap}^\star$ to represent the fact that $\mathrm{Cap}_{31}(2, 25)$ is consumed on a idle processor. In general, note that in the interval between the capacity creation and its depletion there can be intervals of time in which the

capacity is not consumed at all, either because M processors with strictly shorter deadline execute or another capacity with shorter deadline is at the head of the capacity queue. Hence, each capacity job is created either when the generating server job finishes or the capacity consumption rate changes. Furthermore, note that since a capacity job is only created when the corresponding capacity is consumed, the sum of the computation times of all capacity jobs generated by a capacity (c_q, d_q) is in fact equal to c_q.

The importance of the job transformation lies in the fact that the scheduling of I and $I^\star$ are equivalent, as the following lemmas show.

Lemma 6 *The global EDF scheduling of job set $I^\star$ on π (where capacity jobs are arbitrarily given higher priority than server jobs with the same absolute deadline) is equivalent to the M-CASH scheduling of I on π, in the sense that:*

- *if S_{ij} executes consuming its own budget in the M-CASH schedule of I at time t, $S^\star_{ij}$ executes in the EDF schedule of $I^\star$ at time t;*
- *for each server job S_{ij} executing consuming a capacity in the M-CASH schedule of I at time t, a capacity job with deadline less than or equal to d^S_{ij} executes in the EDF schedule of $I^\star$ at time t.*

Proof.
The proof follows directly from Definition 3, since for each capacity Cap_{ij} and each interval $[t'_l, t_l]$, $M - \#V$ capacity jobs are executing instead of the $E - V$ server jobs that execute consuming a capacity, and therefore all capacity jobs created at t_{l-1} finish exactly at t_l (note that since in the M-CASH schedule of I the capacity is not consumes in $[t_{l-1}, t'_l]$, the capacity jobs in $I^\star$ are preempted by either server jobs or capacity jobs with smaller deadline). □

Lemma 7 *If $I^\star$ is feasibly scheduled by global EDF on π, then no job in I misses its deadline when scheduled by M-CASH on π.*

Proof.
Whenever a job S_{ij} executes in the M-CASH schedule of I, either job $S^\star_{ij}$ or a capacity job with earlier or equal deadline than d^S_{ij} must execute in the EDF schedule of $I^\star$ by Lemma 6. Since all jobs in $I^\star$ meet their deadline, it thus follows that job S_{ij} must meet its deadline as well (note that the computation time c^S_{ij} of S_{ij} is equal to the computation time $c^\star_{ij}$ of $S^\star_{ij}$ plus the amount of capacity consumed by S_{ij} by definition). □

Instead of proving that the M-CASH scheduling of a job set I is correct under the same conditions $U_\pi \leq M - u_\pi(M-1), u_\pi \leq 1$ of Theorem 4, we can thus prove that set $I^\star$ can be feasibly scheduled by global EDF under those conditions. In particular, we would like to reuse the same framework as in Section 2: first we show that set $I^\star$ is feasible on a uniform multiprocessor with speeds $s_i = U_i$, then using Theorem 2 we conclude that $I^\star$ is schedulable by EDF on identical multiprocessor π and therefore due to Lemma 7 I is also schedulable by M-CASH. Unfortunately, this approach does not directly work since it is not always possible to schedule $I^\star$ on such uniform multiprocessor. To understand the problem, consider the first server job S_{i1} of server S_i and suppose that it completes at time t leaving a capacity $\mathrm{Cap}_{i1}(c_q, d_q = d^S_{i1})$ in the M-CASH queue; furthermore, suppose that only one capacity job in $\mathrm{Cap}^\star_{ij}$ is generated. Then we would like to schedule both the transformed job $S^\star_{i1}$ and $\mathrm{Cap}^\star_{ij}$ on a dedicated uniform processor with speed $s_i = U_i$. However, if the condition $(d_q - t)U_i < c_q$ holds, it is easy to see that the capacity job will miss its deadline. As an example, consider server $S_i(Q_i = 4, T_i = 8)$ and suppose that its server job $S_{i1}(a^S_{i1} = 0, d^S_{i1} = 8)$ executes for two times units in $[4, 6]$, leaving a capacity $C_{i1}(c_q = 2, d_q = 8)$ in the M-CASH queue. Then the corresponding capacity job with $a^\star_{i11} = 6, d^\star_{i11} = 8, c^\star_{i11} = 2$ has a demand equal to $\frac{c^\star}{d^\star_{i11} - a^\star_{i11}} = 1 > U_i = 0.5$; hence, it is not schedulable on a uniform processor with speed $s_i = U_i$.

We can solve the problem by slightly modifying the analysis. First, we introduce a second job transformation from $I^\star$ to a set I' that is schedulable on a uniform multiprocessor with $s_i = U_i$. Then, we extend Theorem 2 to show that if I' is schedulable on the uniform platform than $I^\star$ is schedulable by global EDF on the identical platform π. Since by Lemma 7 the schedulability of $I^\star$ implies the schedulability of I, we can then conclude that M-CASH is correct. Before formally defining I', let us provide the main intuition behind the job sets $I, I^\star, I'$. $I^\star$ is the comprehensive job set comprised of server jobs $S^\star$ and capacity jobs $\mathrm{Cap}^\star$, which thus represent the execution of the rules of M-CASH "transformed" into an EDF scheduling. The original job set I can be though of as the collection of server jobs from $S^\star$, where each server job is "merged" with all the capacity jobs that it *consumes*; capacity jobs consumed by idle processors are non included. Then, the *serial job transformation* I' is the set of server jobs from $S^\star$, where each server job is "merged" with all the capacity jobs that it *generates*; all capacity jobs are thus merged. Hence, the execution of a serial job S'_{ij} in I' corresponds to either the execution of S_{ij} in I consuming its own budget, or the consumption of Cap_{ij} on any processor. The main intuition is that server jobs S'_{ij} have the same characteristics of M-CBS server jobs, and are therefore schedulable on the uniform platform.

Definition 4 (Serial job transformation) *Consider a set I of server jobs generated by M-CASH system π and its M-CASH job transformation $I^\star$. Then the serial job transfor-*

mation I' of I is comprised by a set of jobs $\{S'_{ij}\}$: for each job $S^\star_{ij}$ in $S^\star$, $S'_{ij} = (a'_{ij} = a^\star_{ij}, d'_{ij} = d^\star_{ij}, c'_{ij} = c^\star_{ij} + c_q\}$ where c_q is the capacity of Cap_{ij} (zero if no capacity is generated by S_{ij}) or equivalently the sum of the computation times of all the associated capacity jobs $\mathrm{Cap}^\star_{ij}$.

Lemma 8 *No job S'_{ij} misses its deadline if S'_i is assigned to a dedicated uniform processor with speed $s_i = U_i$.*

Proof.
Since $c^\star_{ij}$ is equivalent to the amount of budget consumed by S_{ij}, it follows that the computation time of $S'_{ij} : c'_{ij} = c^\star_{ij} + c_q = Q_i$. Since furthermore $a'_{ij} = a^\star_{ij} = a^S_{ij}$ and $d'_{ij} = d^\star_{ij} = d^S_{ij}$, the proof follows directly from Lemma 3. □

Theorem 9 *Consider a uniform multiprocessor system π' with $s_{\pi'} \leq 1$, and let the serial transformation I' be feasible on π'. Then the EDF schedule of $I^\star$ on π (where capacity jobs are arbitrarily given higher priority than server jobs with the same absolute deadline) is feasible if:*

$$M \geq \frac{S_{\pi'} - s_{\pi'}}{1 - s_{\pi'}}$$

Proof sketch.
Consider the set of jobs $S^\star_{ij} \cup \mathrm{Cap}^\star_{ij}$, comprised of the M-CASH transformation of S_{ij} and all the capacity jobs generated by Cap_{ij} (if any). The sum of the execution times of all such jobs is equal to the computation time c'_{ij} of the serial transformed job S'_{ij} and furthermore all considered jobs have the same deadline. Hence, the only difference among the two transformed sets is that while some of the capacity jobs in the M-CASH transformed set can execute in parallel, the execution of S'_{ij} is completely serialized. Therefore, we expect that a multiprocessor system can execute more work on the M-CASH transformed set than on the serial transformed set. Since by Theorem 2 I' is feasible on π, $I^\star$ is thus feasible on π as well.

The complete proof can be found in appendix. □

Using Theorem 9 we can now prove our main theorems, which mirror Theorems 4 and 5 for M-CASH.

Theorem 10 *No M-CASH server job S_{ij} misses its deadline on π under the following conditions:*

$$u_\pi \leq 1;$$

$$U_\pi \leq M - u_\pi(M - 1)$$

Proof.
Consider the set I of all M-CASH generated jobs and the transformed sets $I^\star, I'$. Due to Lemma 8, I' is feasible on a uniform platform π' with $S_{\pi'} = U_\pi$ and $s_{\pi'} = u_\pi$. From $U_\pi \leq M - u_\pi(M-1)$ it follows $M \geq \frac{S_{\pi'} - s_{\pi'}}{1 - s_{\pi'}}$, therefore by Theorem 9 $I^\star$ can be feasibly scheduled by EDF on the identical platform π. Hence, by Lemma 7 no server job in I misses its deadline. □

Theorem 11 *A periodic task τ_i with period and relative deadline T_i and execution time Q_i can be feasibly scheduled by M-CASH server S_i under the conditions of Theorem 10.*

Proof.
Since the computation time of τ_i is equal to Q_i and because of Theorem 10, job τ_{i1} will complete before its deadline leaving a budget $c_i \geq 0$. We can thus reuse the same reasoning as in Theorem 5 to complete the proof. □

As a conclusion to this section, we want to stress that the proof of correctness is strongly dependant on the bound being used. In particular, the assumptions made on the proof sketch of Theorem 9 only hold because we can prove that no scheduling anomaly is present under the assumptions of Theorem 9.

4. Experimental evaluation

Both the M-CBS (in its original version detailed in [5]) and the M-CASH algorithm have been implemented in the real-time simulator RTSIM [12] to measure the performance of our resource reclaiming scheme through several synthetic simulations. In particular, we have considered a system of $M = 4$ processors with $N = 20$ tasks: $\tau_1, \ldots, \tau_{16}$ are periodic hard tasks, each served by a server with maximum budget equal to the task worst-case computation time; $\tau_{17}, \ldots, \tau_{20}$ are periodic soft tasks that can experience overload conditions. All task periods are chosen to be uniformly distributed in the interval $[100, 5000]$. In each experiment, the performance of the server algorithm is evaluated by computing the average tardiness and the normalized response time for all soft tasks over a run of $500,000$ time units. Given a soft job $\tau_{ij}, 17 \leq i \leq 20$, the job tardiness Tard_{ij} and normalized response time Resp_{ij} are defined as follows:

$$\mathrm{Tard}_{ij} = \frac{\max(f_{ij} - d_{ij}, 0)}{d_{ij} - a_{ij}}$$

$$\mathrm{Resp}_{ij} = \frac{f_{ij} - a_{ij}}{c_{ij}}$$

where f_{ij} is the finishing time of τ_{ij}. The global tardiness and normalized response time are computed by averaging over all soft jobs executed in the run.

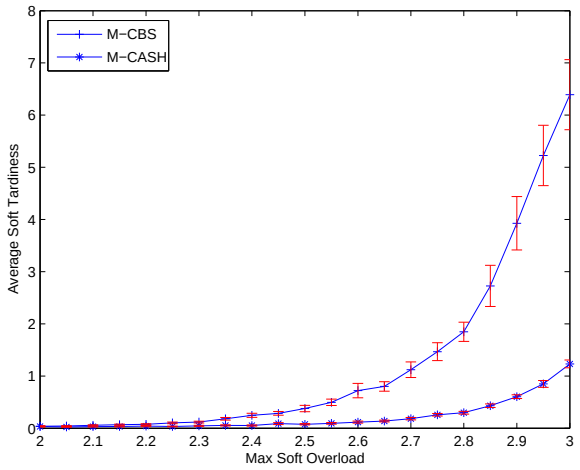


Figure 4. Average Tardiness for $\alpha = 0.7$

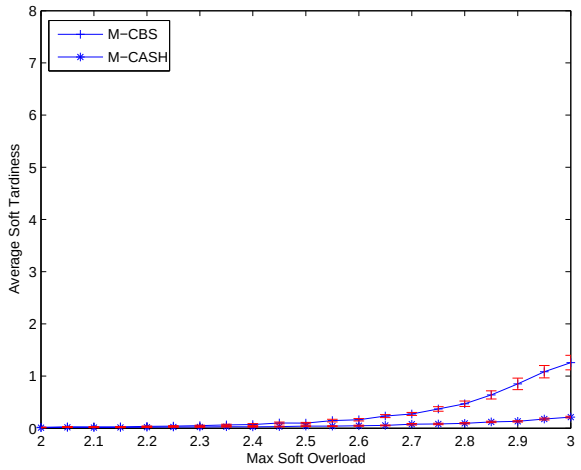


Figure 5. Average Tardiness for $\alpha = 0.5$

Each soft task is assigned a server bandwidth $U_S = 0.3$, while hard tasks are constrained to utilizations no greater than U_S. Hence, following Theorems 5,11, we can guarantee hard task schedulability if $U_\pi \leq M-(M-1)U_S = 3.1$. In order to use the maximum guaranteed bandwidth, we can thus impose a total bandwidth for hard tasks equal to the computed bound of 3.1 minus the soft task bandwidth, i.e. $\sum_{i=1}^{16} U_H = M-(M-1)U_S-\sum_{i=17}^{20} U_S = 1.9$. Hard task utilizations are chosen to be uniformly distributed with sum equal to 1.9, and worst case computation times are computed based on task periods and utilizations.

We have analyzed several task set scenarios by varying two critical parameters. The first considered parameter α is the minimum relative computation time for all hard tasks, or formally:

$$\alpha = \frac{c_i^{\min}}{c_i} \; \forall i, 1 \leq i \leq 16$$

where the computation time of any job τ_{ij} is chosen to be uniformly distributed in $[\alpha c_i, c_i]$. Hence, α provides a measure of the amount of bandwidth left free by early completing hard jobs. The second parameter γ is the maximum soft task overload, defined as:

$$\gamma = \frac{c_i^{\max}}{Q_i} \; \forall i, 17 \leq i \leq 20$$

where each soft job τ_{ij} can execute for at most $c_{ij} = \gamma Q_i$ time units. We present two sets of experiments: in the first one, we fix α to a predetermined value and we analyze the average tardiness and response time varying γ, while in the second set of experiments we fix γ and vary α instead.

In Figures 4 and 5 we plot the results for the first set of experiments, with $\alpha = 0.7$ and $\alpha = 0.5$ respectively and γ varying from 2 to 3. Due to space constraints and since the response time graphs exhibit similar trends, we only plot the average tardiness with 95% confidence intervals. As expected, the tardiness is smaller for the experiments with $\alpha = 0.5$, since hard tasks execute for less time thus leaving more bandwidth for soft task execution. In both experiments, M-CASH significantly outperforms M-CBS by reducing the average tardiness up to 6 times. Furthermore, note that in both cases the tardiness is very close to zero for $\alpha \leq 2.2$. Since this value corresponds to more than a 100% increase in soft task computation time in the worst case, we could expect to see larger tardiness values. Indeed, this would be the case on a uniprocessor EDF platform where the guaranteed bandwidth is equal to 1. However, following Theorems 5,11 we can only guarantee hard task execution for a total server bandwidth equal to 3.1, which means that the residual bandwidth of $4-3.1 = 0.9$ available on the four processor system can not be assigned to any server. In fact, note that even for $\alpha = 2$ the tardiness is not zero, signifying that in the worst case the residual bandwidth can not be used. However, in the average case overruning servers are indeed able to exploit the residual bandwidth, thus lowering the average tardiness significantly.

In Figure 6 we show the second set of experiments where the maximum overload is fixed to a value $\gamma = 2.7$ and α is varied in the interval $[0.3, 0.9]$. Since no capacity is left free by hard tasks for $\gamma = 1$, the graphs tend to explode for high values of γ. Again, note that while M-CBS shows significant tardiness even for small values of γ, M-CASH is able to obtain decidedly better performance, achieving extremely small tardiness values for $\alpha \leq 0.7$.

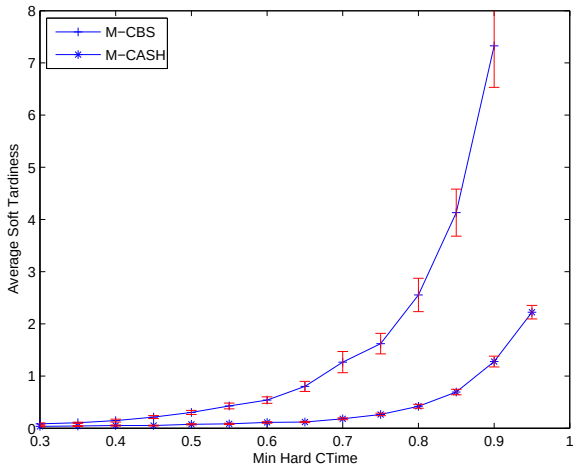


Figure 6. Average Tardiness for $\gamma = 2.7$

5. Conclusions

In this paper, we have introduced and proved correct M-CASH, a new scheduling algorithm that augments the M-CBS server for resource reservation in multiprocessor system with a powerful resource reservation mechanism. Simulations based on the RTSIM platform show that M-CASH is able to greatly improve the performance of soft tasks in the presence of overruns with respect to M-CBS. Hence, we conclude that the implementation of M-CASH is in fact beneficial for systems of mixed hard-soft real-time tasks executed on an identical multiprocessor platform.

As future work, we plan to extend our resource reclaiming scheme in two directions. First of all, as detailed in Section 3.1 the correctness of M-CASH depends on the schedulability bound being used. Although the considered bound is strict, it is still pessimistic since it relies on task utilizations only and not on all task parameters. It would be interesting to extend our analysis to less pessimistic bounds such as the one in [6]. Furthermore, note that independentely from the bound, the guaranteed bandwidth is always less than the number of available processors. Hence, there is some residual capacity in the system that is not assigned to any server and is not consider by M-CASH; this capacity can still be used by overruning tasks possibly causing continuous deadline postponement (typically known as deadline aging). We are currently developing an extension to M-CASH that is able to avoid this problem.

Finally, while in this work we have been concerned with global scheduling only, partitioned scheduling, where tasks are statically assigned to processors, is also popular for real-time multiprocessor systems. While the static nature of partitioned scheduling makes it difficult to directly apply the M-CASH scheme, we think that restricting migration to overruning jobs should still allow for efficient inter-processor resource reclaiming.

References

[1] L. Abeni. Server mechanisms for multimedia applications. Technical Report RETIS TR98-01, Scuola Superiore S. Anna, 1998.

[2] L. Abeni and G.Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, Madrid, Spain, december 1998. IEEE.

[3] S. Baruah, N. Cohen, C. Plaxton, and D. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 6, 1996.

[4] S. Baruah and N. Fisher. The partitioned multiprocessor scheduling of sporadic task systems. In *Proceedings of the* 26^{th} *IEEE Real-Time Systems Symposium (RTSS'05)*, Miami, Florida, 2005.

[5] S. Baruah, J. Goossens, and G. Lipari. Integrating constant-bandwidth servers upon multiprocessor platforms. In *Proceedings of* 8^{th} *Real-Time Application Symposium (RTAS'02)*, 2002.

[6] M. Bertogna, M. Cirinei, and G. Lipari. Improved schedulability analysis of edf on multiprocessor platforms. In *Proceedings of* 17^{th} *Euromicro Conference on Real-Time Systems (ECRTS'05)*, pages 209–218, Palma de Mallorca, Spain, 2002.

[7] M. Caccamo, G. Buttazzo, and L. Sha. Capacity sharing for overrun control. In *Proceedings of the IEEE Real-Time Systems Symposium*, Orlando, Florida, December 2000.

[8] M. Caccamo, G. Buttazzo, and D. Thomas. Effi cient reclaiming in reservation-based real-time systems with variable execution times. *IEEE Transactions on Computers*, 54(2), February 2005.

[9] J. Goossens, S. Funk, and S. Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-Time Systems Journal*, 25:187–205, 2003.

[10] G. Lipari and S. Baruah. Greedy reclamation of unused bandwidth in constant-bandwidth servers. In *Proceedings of* 12^{th} *Euromicro Conference on Real-Time Systems (ECRTS'00)*, 2000.

[11] L. Marzario, G. Lipari, P. Balbastre, and A. Crespo. Iris: a new reclaiming algorithm for server-based real-time systems. In *Proceedings of the* 10^{th} *Real-Time Application Symposium (RTAS'04)*, Toronto, Canada, May 2004.

[12] Scuola Superiore Sant'Anna. *Real-Time System SIMulator (RTSIM)*. http://rtsim.sssup.it/.

A. Proof of Theorem 9

The proof closely follows the one of Theorem 2 in [9]. To ease the notation, define $\bar{S}^\star_{ij} = S^\star_{ij} \cup \mathrm{Cap}^\star_{ij}$. Furthermore, let I'_k be the set of the k shorter deadline jobs in I' and similarly $I^\star_k$ be the union set of the k sets $\bar{S}^\star_{ij}$ with shorter deadline (note that all jobs in $\bar{S}^\star_{ij}$ have the same deadline). Finally, let $W(A, \pi', I'_k, t)$ be the amount of time that jobs in I'_k execute on π' in the interval $[0, t]$ when scheduled by a scheduling algorithm A; similarly $W(\mathrm{EDF}, \pi, I^\star_k, t)$ is the amount of time that jobs in $I^\star_k$ execute on π in $[0, t]$ when scheduled by global EDF.

We begin by proving the following lemma:

Lemma 12 *Under the assumptions of Theorem 9, for any possible scheduling algorithm A:*

$$\forall k \geq 0, \forall t \geq 0 : W(\mathrm{EDF}, \pi, I^\star_k, t) \geq W(A, \pi', I'_k, t)$$

Proof.
By contradiction. Suppose that it is not true, in the sense that there is a instant of time t_0 such that the amount of work done on I'_k on π' is greater than the amount of work done on $I^\star_k$ on π. Then there must be at least one job $\bar{S}^\star_{ij}, S'_{ij}$ such that:

$$W(\mathrm{EDF}, \pi, \{\bar{S}^\star_{ij}\}, t_0) < W(A, \pi', \{S'_{ij}\}, t_0)$$

Let S'_{ij} be such job with earliest activation time. Then since $a^\star_{ij} = a'_{ij}$, it follows that

$$W(\mathrm{EDF}, \pi, a^\star_{ij}, t_0) \geq W(A, \pi', a'_{ij}, t_0)$$

and hence the work done on $I^\star_k$ over the interval $[a'_{ij}, t_0]$ must be less than the work done on I'_k over the same interval. Now denote with x the amount of time over the interval $[a'_{ij}, t_0]$ during which all M processors are busy on π; conversely, let $y = t_0 - a'_{ij} - x$ be the amount of time during which at least one processor is idle on π. We make the following two observations.

- First note that since the amount of work done on $\bar{S}^\star_{ij}$ is strictly less than the amount of work done on S'_{ij} and the total computation time of jobs in $\bar{S}^\star_{ij}$ is equal to c'_{ij}, it follows that some job in $\bar{S}^\star_{ij}$ has not finished yet at t_0. Furthermore, since $I^\star_k$ is the union set of the k sets $\bar{S}^\star_{ij}$ with shorter deadline, the EDF schedule of $I^\star_k$ is not affected by jobs in $I^\star - I^\star_k$. Hence, following Definition 3 and Lemma 6 each capacity job in $\bar{S}^\star_{ij}$ is activated either when $S^\star_{ij}$ finishes or a previous capacity job in $\bar{S}^\star_{ij}$ finishes, which means that there is always at least one active job (i.e. either executing or in the ready queue) of $\bar{S}^\star_{ij}$ in the interval $[a'_{ij}, t_0]$. Therefore, jobs in $\bar{S}^\star_{ij}$ must execute for at least y time units in $[a'_{ij}, t_0]$, while S'_{ij} can not execute for more than $s_{\pi'}(x + y)$ time units. It thus follows:

$$s_{\pi'}(x + y) > y \tag{1}$$

- Since at least one job of $\bar{S}^\star_{ij}$ is active in $[a'_{ij}, t_0]$, it follows that the amount of time $I^\star_k$ executes in $[a'_{ij}, t_0]$ is at least $Mx + y$, while the amount of time I'_k executes in the same interval is at most $S_{\pi'}(x + y)$. It thus follows:

$$S_{\pi'}(x + y) > Mx + y \tag{2}$$

By adding $M - 1$ times Inequality 1 to Inequality 2, we get [9]:

$$\frac{S_{\pi'} - s_{\pi'}}{1 - s_{\pi'}} > M$$

which contradicts our assumption. □

Using Lemma 12 we can prove the main theorem by induction on the value k in $I^\star_k$.

Base Case. Since $I^\star_0$ is the empty set, it is clearly feasible on π.

Induction Step. Assuming that $I^\star_k$ can be feasibly scheduled by global EDF on π, we need to prove that $I^\star_{k+1}$ is schedulable on π as well. Again, note that the schedule of $I^\star_k$ is not affected by jobs in $I^\star - I^\star_k$; hence, all jobs in $I^\star_k$ meet their deadlines in the schedule of $I^\star_{k+1}$ by the induction hypothesis. It therefore remains to prove that jobs in $I^\star_{k+1} - I^\star_k$ meet their deadlines; assume, without loss of generality, that they are the jobs in $\bar{S}^\star_{ij}$.

Now consider the schedule of I'_{k+1} on π'. Since I' is feasible on π', it follows that I'_{k+1} must be feasible as well, in the sense that there exists a scheduling algorithm OPT on π' which produces a feasible schedule for I'_{k+1}. Therefore by Lemma 12:

$$W(\mathrm{EDF}, \pi, I^\star_{k+1}, d^\star_{ij}) \geq W(\mathrm{OPT}, \pi', I'_{k+1}, d'_{ij}) =$$

$$= \sum_{\forall S'_{pq} \in I'_{k+1}} c'_{pq}$$

But since no job in $I^\star_{k+1}$ has deadline greater than $d^\star_{ij}$ and the sum of the computation times of jobs in $I^\star_{k+1}$ is equal to the sum for jobs in I'_{k+1}, if follows that all jobs in $I^\star_{k+1}$ must complete before $d^\star_{ij}$. Hence, $\bar{S}^\star_{ij}$ is schedulable as well. Since all jobs in $I^\star_{k+1}$ meet their deadlines, the theorem follows.